Class 1: Introduction to DevOps

Session Overview

- Understand the fundamental concepts and principles of DevOps.
- Recognize the key differences between traditional software development practices and DevOps.
- Identify the benefits of adopting a DevOps culture in an organization.
- Get an overview of the DevOps toolchain and its significance.

Prerequisites

- Basic understanding of software development and operations.
- Familiarity with Agile methodologies is a plus.

Introduction

This class serves as the foundation for understanding DevOps—a set of practices aimed at bridging the gap between software development (Dev) and IT operations (Ops). The primary goal of DevOps is to improve collaboration, enhance automation, and ensure the continuous delivery of high-quality software. This session will introduce the core principles, benefits, and tools associated with DevOps, setting the stage for the hands-on practices that will be explored throughout the course.

Understanding DevOps

DevOps is a cultural and professional movement that emphasizes collaboration and communication between software developers and IT professionals. It also focuses on automating the processes of software delivery and infrastructure changes.

Significance of DevOps:

- Accelerates software delivery: Faster time to market through streamlined processes.
- Improves collaboration and communication: Breaks down silos between teams.
- Enables continuous feedback loops: Ensures consistent improvement and quick issue resolution.
- Reduces time to market: Facilitates quicker responses to market demands and customer feedback.

DevOps Principles and Practices

Core Principles:

1. Continuous Integration (CI): Integrating code into a shared repository multiple times a day to detect issues early.
2. Continuous Delivery (CD): Ensuring that code is always in a deployable state.
3. Continuous Deployment: Automating the deployment of code to production without manual intervention.

Key Practices:

- **Automation**: Automating repetitive tasks such as testing, deployment, and infrastructure provisioning.
- **Collaboration**: Enhancing collaboration between development and operations teams to work more efficiently.
- **Measurement**: Monitoring and measuring the performance of applications and infrastructure to drive improvements.

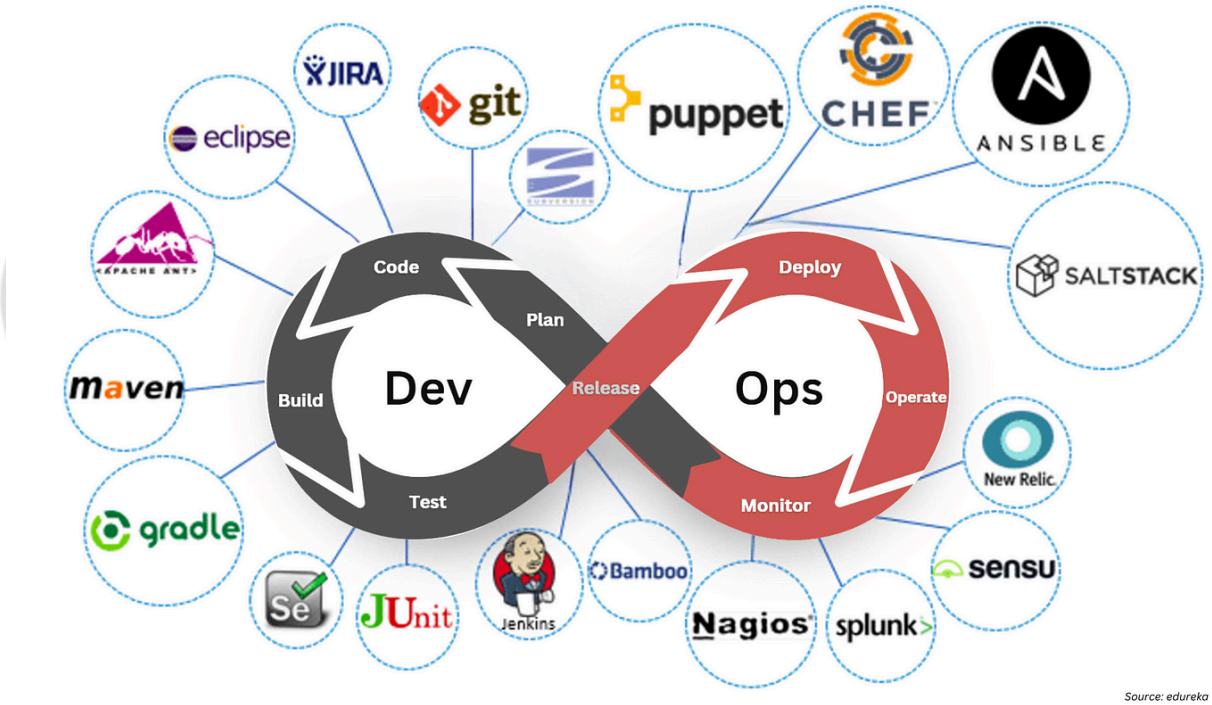
Benefits of DevOps

- **Improved Deployment Frequency**: Shorter development cycles lead to more frequent releases.
- **Faster Time to Market**: Quicker responses to market demands and customer feedback.
- **Lower Failure Rate of New Releases**: Automation reduces human errors, ensuring higher-quality releases.
- **Shorter Lead Time Between Fixes**: Issues are identified and resolved faster, thanks to continuous monitoring.
- **Enhanced Collaboration**: Breaks down silos between teams, fostering a collaborative environment.

DevOps Lifecycle

The DevOps lifecycle represents an iterative process that integrates development and operations teams to improve collaboration, streamline workflows, and enhance the efficiency of software delivery. This lifecycle is continuous and emphasizes the automation of processes, feedback loops, and iterative improvement. Below is a detailed breakdown of the DevOps lifecycle:

codingninjas



1. **Planning**
    - Objective: Define software features, requirements, and system architecture.
    - Activities:
        - Collaborate with stakeholders to gather and prioritize requirements.
        - Create a roadmap and plan for feature development and releases.
        - Use tools like Jira, Trello, or Azure DevOps for task tracking and project management via agile methodology.
    - Outcome: A clear plan outlining what needs to be developed, how it will be implemented, and the timelines for delivery.

Agile Methodology

- Break down the product backlog into smaller tasks within a sprint.
- Conduct daily standups to track progress and remove blockers.
- Perform sprint planning to define the scope and goals for the sprint.
- Use tools like Jira, Trello, or Azure DevOps for sprint tracking and task management.
- Conduct sprint reviews and retrospectives to assess the work done and identify improvements.

2. **Development**
    - Objective: Write, test, and review code based on the planning phase.
    - Activities:
        - Coding: Developers write code using version control systems like Git.

- Code Reviews: Peer reviews to ensure code quality and adherence to standards.
- Unit Testing: Automated testing to validate the functionality of individual code units.
- Tools: IDEs (e.g., Visual Studio Code, IntelliJ IDEA), Git for version control.

- Outcome: High-quality, tested code ready for integration.

3. **Continuous Integration (CI)**
   - Objective: Integrate code changes frequently, allowing for early detection and resolution of issues.
   - Activities:
     - Automated Builds: Code changes are automatically built and compiled.
     - Automated Testing: Run automated tests to verify the functionality of the integrated code. Test Creation using Selenium or jUnit.
     - Tools: Jenkins, Travis CI, CircleCI.
   - Outcome: A unified codebase that is continuously updated and tested, ensuring stability and quality.

4. **Continuous Delivery (CD)**
   - Objective: Ensure that the code is always in a deployable state.
   - Activities:
     - Staging Environment: Deploy the code to a staging environment that mirrors production.
     - User Acceptance Testing (UAT): Allow stakeholders to test the application in a staging environment.
     - Tools: Jenkins, GitLab CI/CD, Spinnaker.
   - Outcome: Code that can be reliably deployed to production at any time.

5. **Continuous Deployment**
   - Objective: Automate the deployment of code to production.
   - Activities:
     - Automated Deployment: Automatically deploy code to production after passing all tests.
     - Deployment Strategies: Implement strategies like Blue-Green Deployment, Canary Releases, or Rolling Deployments to ensure smooth updates, Rollbacks.
     - Tools: Kubernetes, Docker, Ansible, Terraform.
   - Outcome: Frequent, reliable updates to production without manual intervention.

6. **Operations**
   - Objective: Monitor, manage, and maintain the software in the production environment.
   - Activities:
     - Monitoring: Use monitoring tools to track system performance, errors, and user behavior.
     - Logging: Collect and analyze logs to identify issues and improve system performance.

- Incident Management: Detect and resolve production issues quickly.
- Tools: Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), Splunk.

- Outcome: A stable and reliable production environment with proactive monitoring and quick incident resolution.

7. **Continuous Feedback**
   - Objective: Gather feedback from end-users and stakeholders to improve the product.
   - Activities:
     - User Feedback: Collect feedback through surveys, support tickets, and direct communication.
     - Performance Metrics: Analyze metrics like application performance, user engagement, and error rates.
     - Post-Mortems: Conduct post-incident reviews to identify root causes and implement improvements.
     - Tools: Feedback tools like SurveyMonkey, performance analytics tools like New Relic.
   - Outcome: A feedback loop that informs future development cycles and helps improve the product and processes.

8. **Continuous Improvement**
   - Objective: Iterate and improve the entire DevOps process.
   - Activities:
     - Process Optimization: Regularly review and optimize workflows, tools, and practices.
     - Automation Enhancement: Identify new areas for automation to reduce manual effort and errors, increases developer productivity.
     - Learning and Development: Encourage continuous learning and skill development among team members.
     - Tools: Retrospectives, Kaizen events, process management tools like LeanKit.
   - Outcome: An evolving DevOps culture that continually enhances efficiency, quality, and collaboration.

Infrastructure as Code (IaC):

- Automates infrastructure provisioning and management through code.
- Ensures consistency and repeatability across environments.
- Uses tools like Terraform, Ansible, or AWS CloudFormation.
- Allows infrastructure to be version-controlled and easily modified.
- Enhances collaboration between development and operations teams.
- Reduces manual errors and accelerates deployment processes.
- Supports scalability and agility in infrastructure management.

Setting up a DevOps Environment

Introduction to Automation Using Jenkins

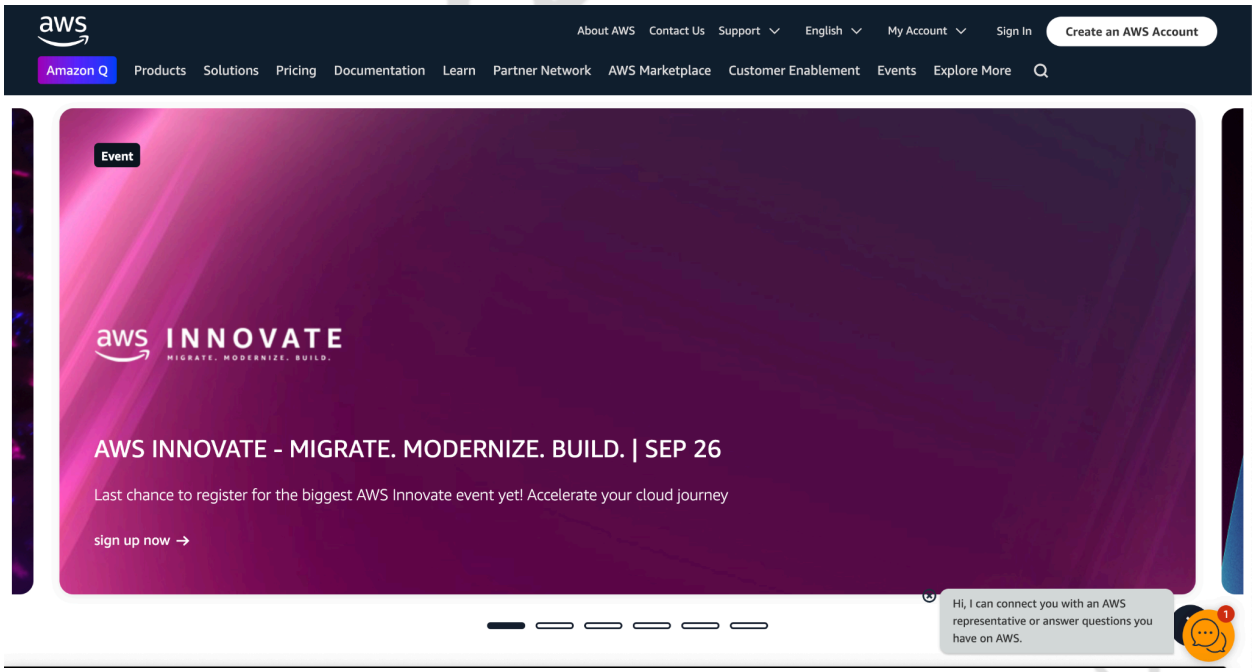
Automation is a critical component of the DevOps process, enabling teams to reduce manual effort, minimize errors, and accelerate the software delivery pipeline. Jenkins is a widely-used open-source automation server that helps automate various stages of the software development lifecycle, such as building, testing, and deploying code. By automating these repetitive tasks, Jenkins allows teams to focus on more strategic and creative aspects of development, leading to faster and more reliable releases.

Introduction to Containerization Using Docker

Containerization is a key technology in modern DevOps practices, allowing developers to package applications and their dependencies into isolated containers. Docker is a popular platform that facilitates containerization, enabling consistent environments across development, testing, and production. Docker containers are lightweight, portable, and can be easily deployed across different environments, making them ideal for scalable and efficient software deployment.

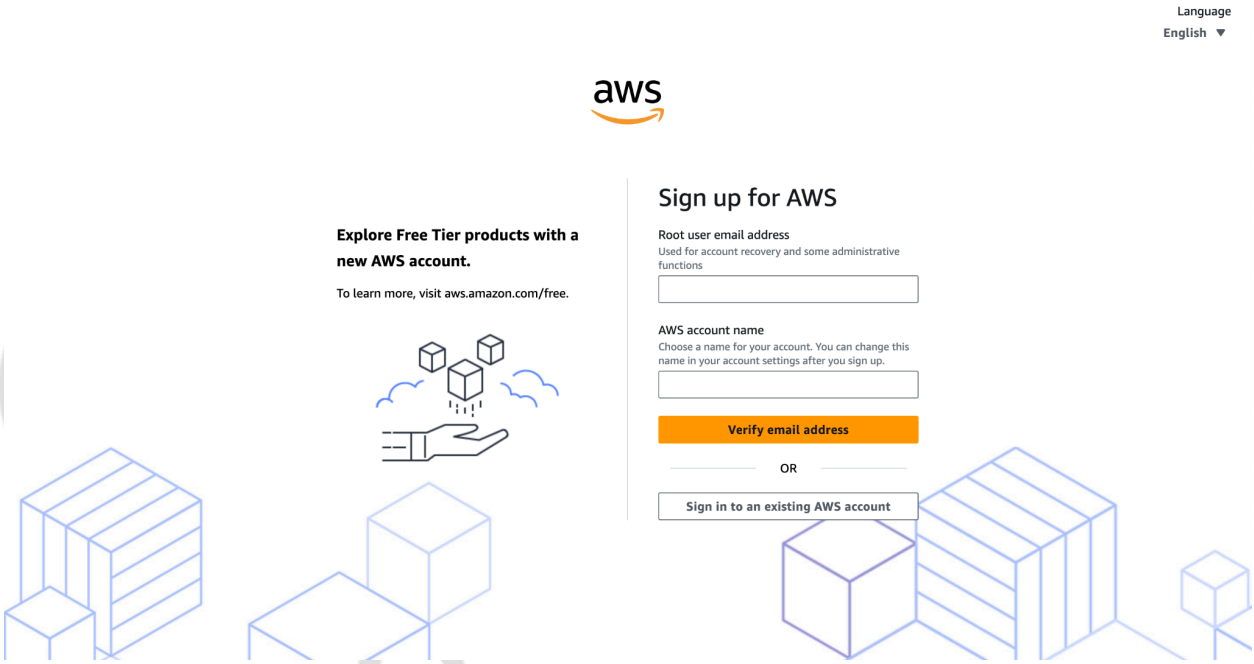
AWS Account Setup

Click on this link to access website https://aws.amazon.com/

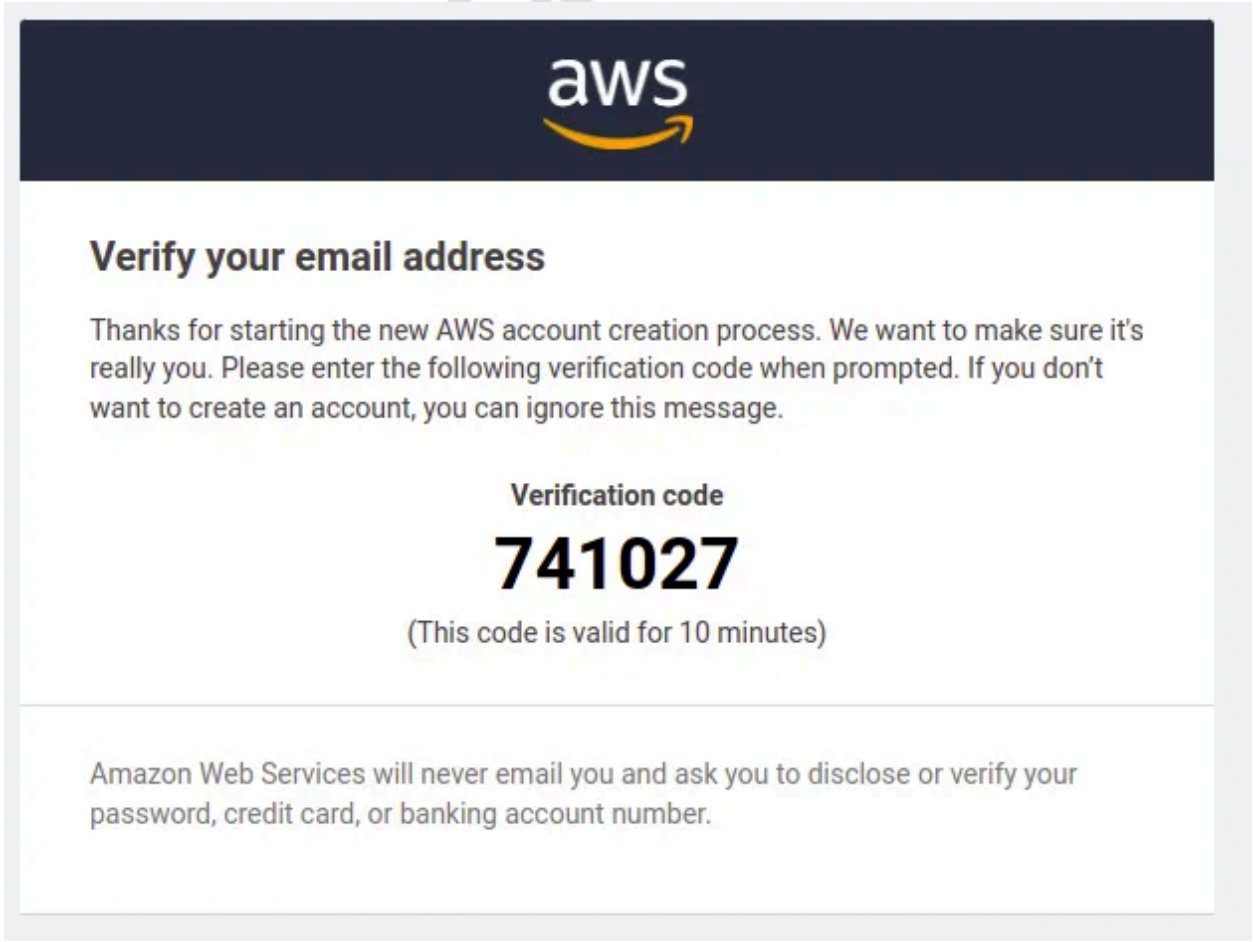


Click on the button on the top right **Create an AWS Account**

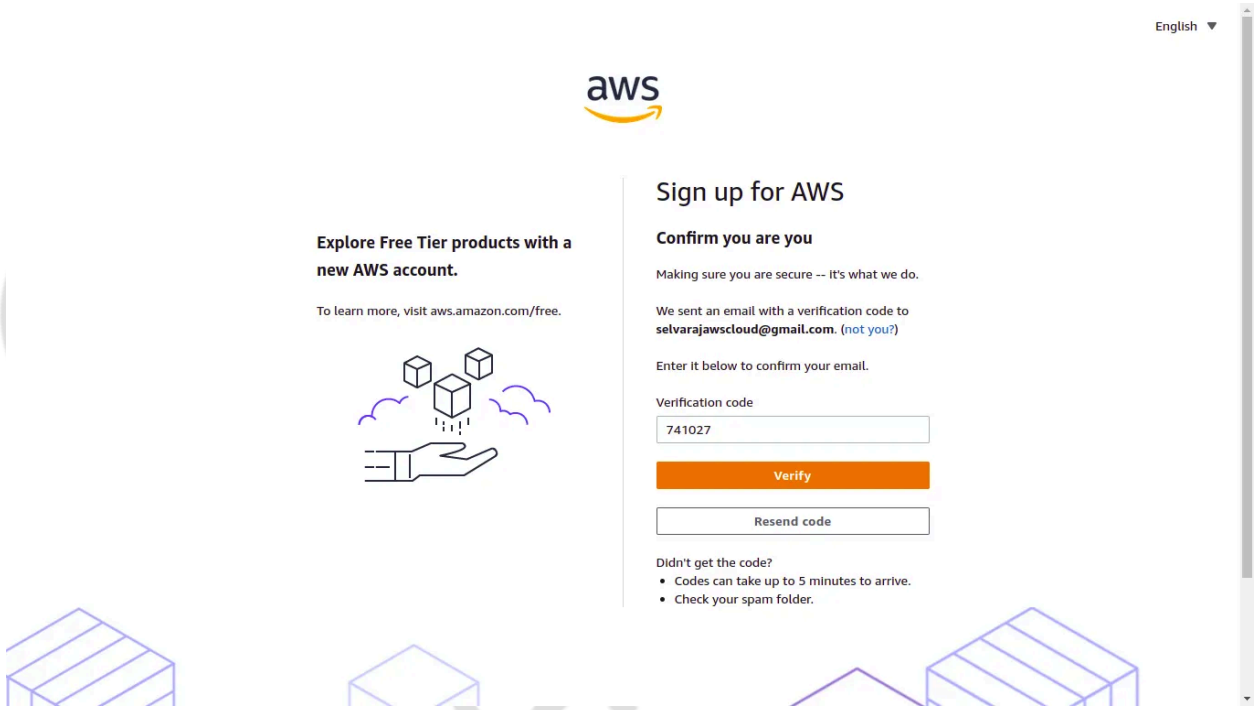
Now enter valid details like email address, password, AWS account name. Now click on **Verify email address.**



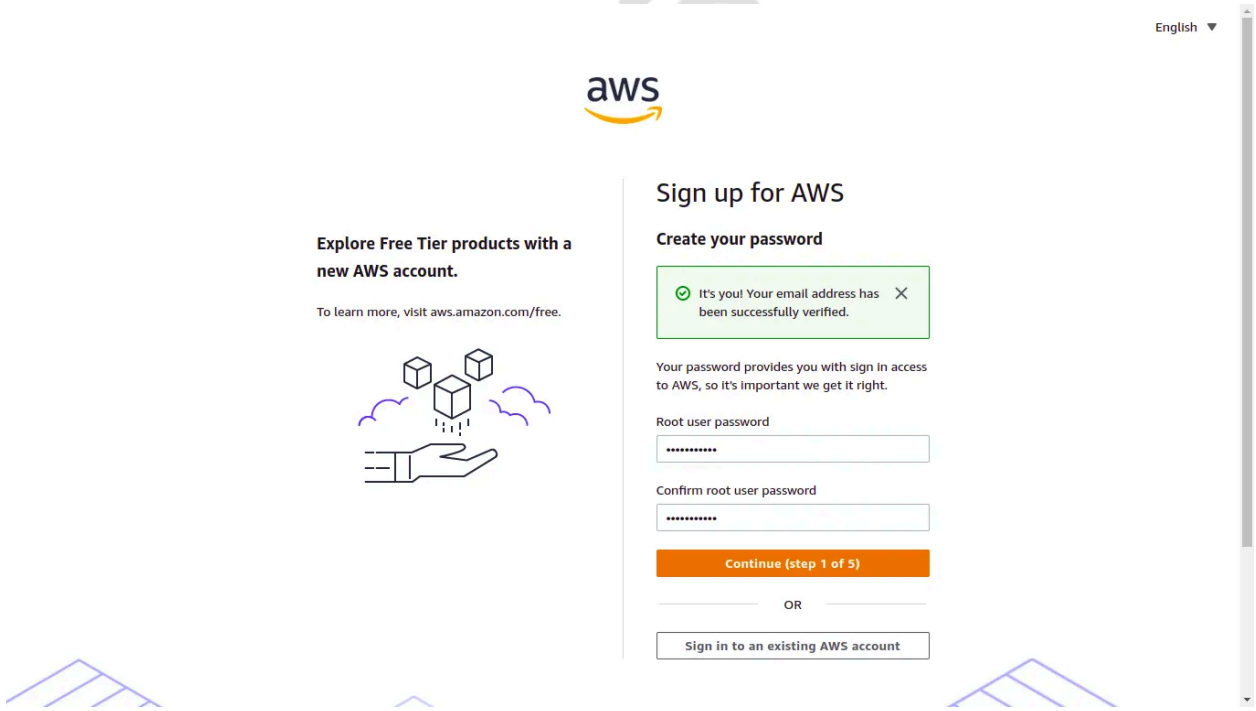
Check the respective email and copy the code to paste it in the website.



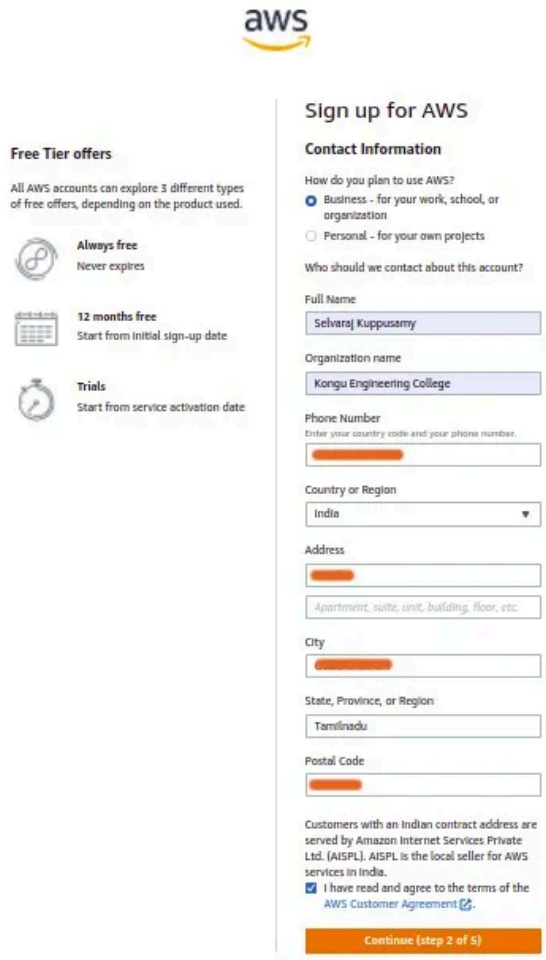
Past the code in the **verify code section** and click **verify**



Email address has been successfully registered. Now set **root user password** then click on **continue**



Now the registration process will ask you your personal details. You should enter valid information.

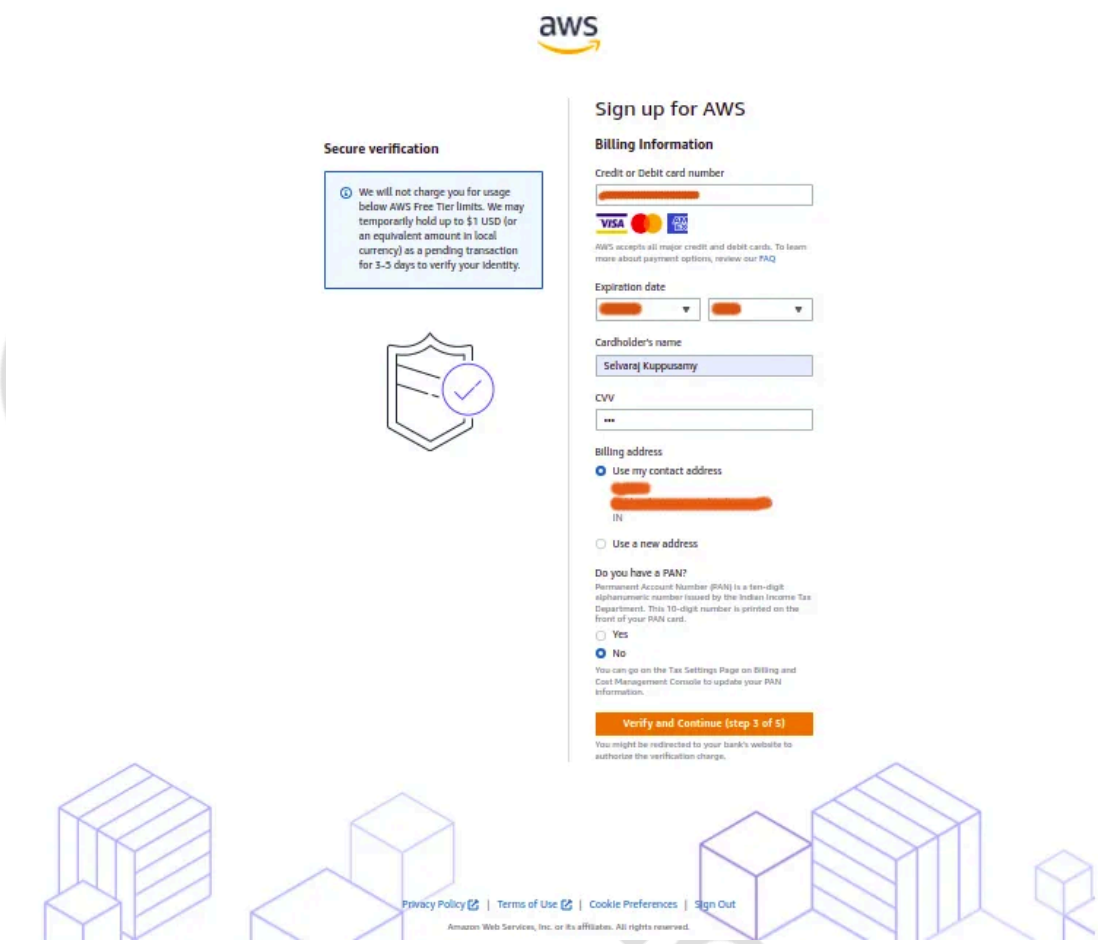


Now, you will provide your payment information. Don't worry about charges right now.

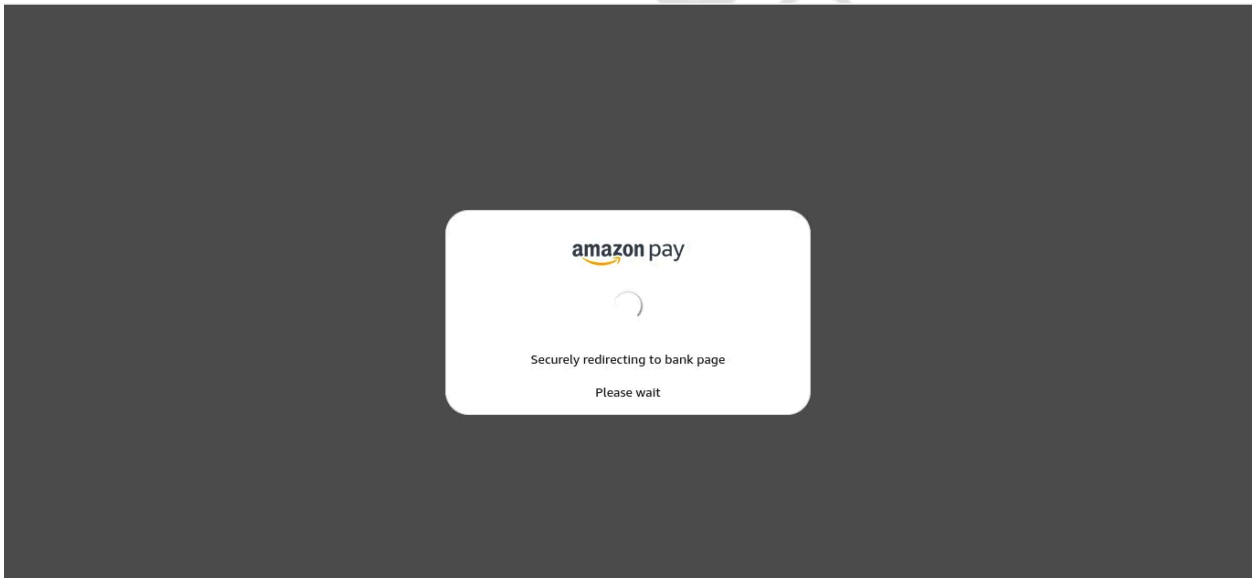
According to AWS: Please type your payment information so, we can verify your identity. We will not charge you unless your usage exceeds the [AWS Free Tier Limits]

In fact, depending on your credit card or debit card company, may be AWS could charge you about **USD $1.00** to check if it is a valid credit card or debit card. They usually undo this charge later.

Click **verify and continue.**



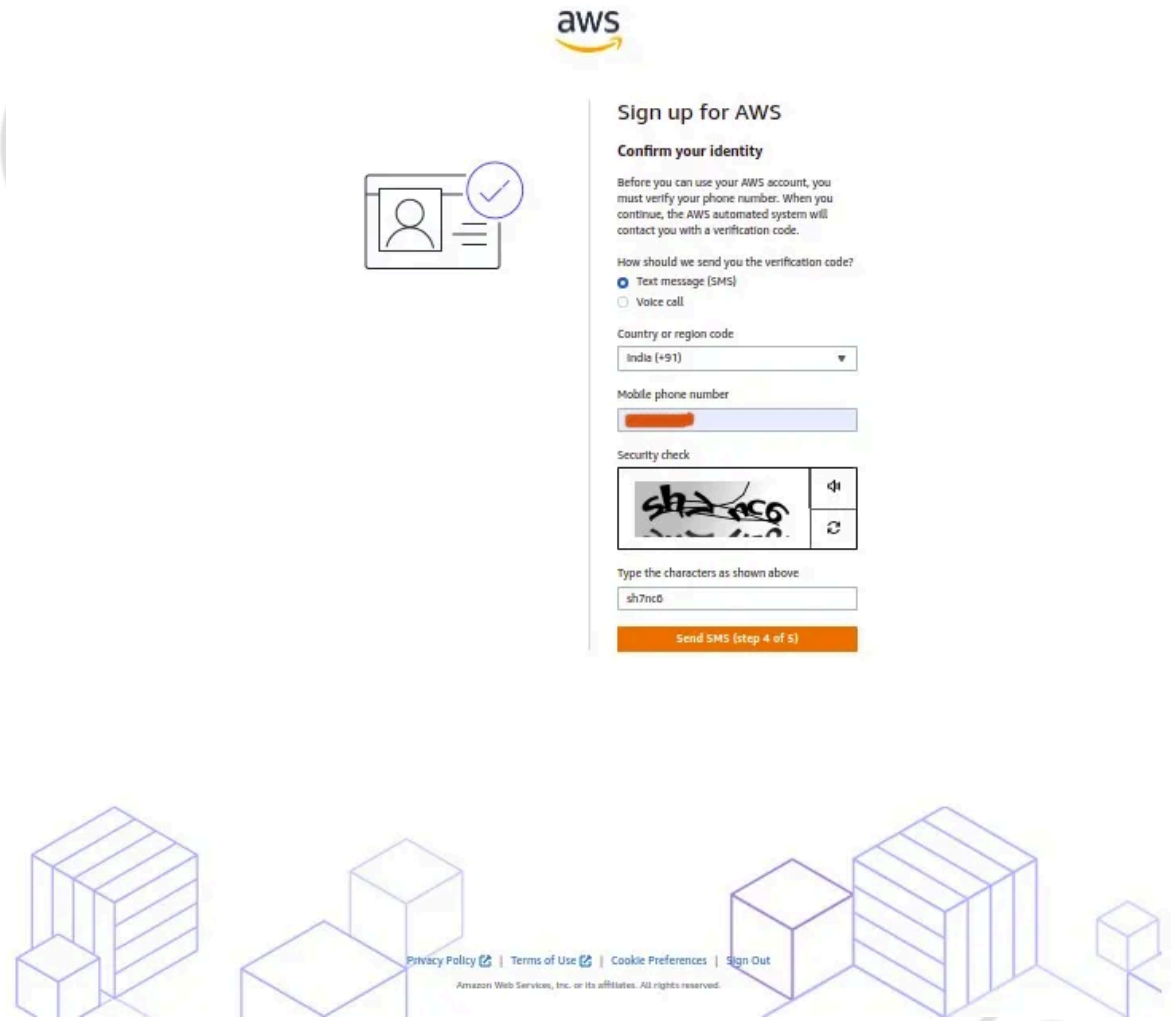
Processing the payment process.



One Time Password will send the respective phone number.

Now, Enter the one time password in the respective **bank** page. and then submit.

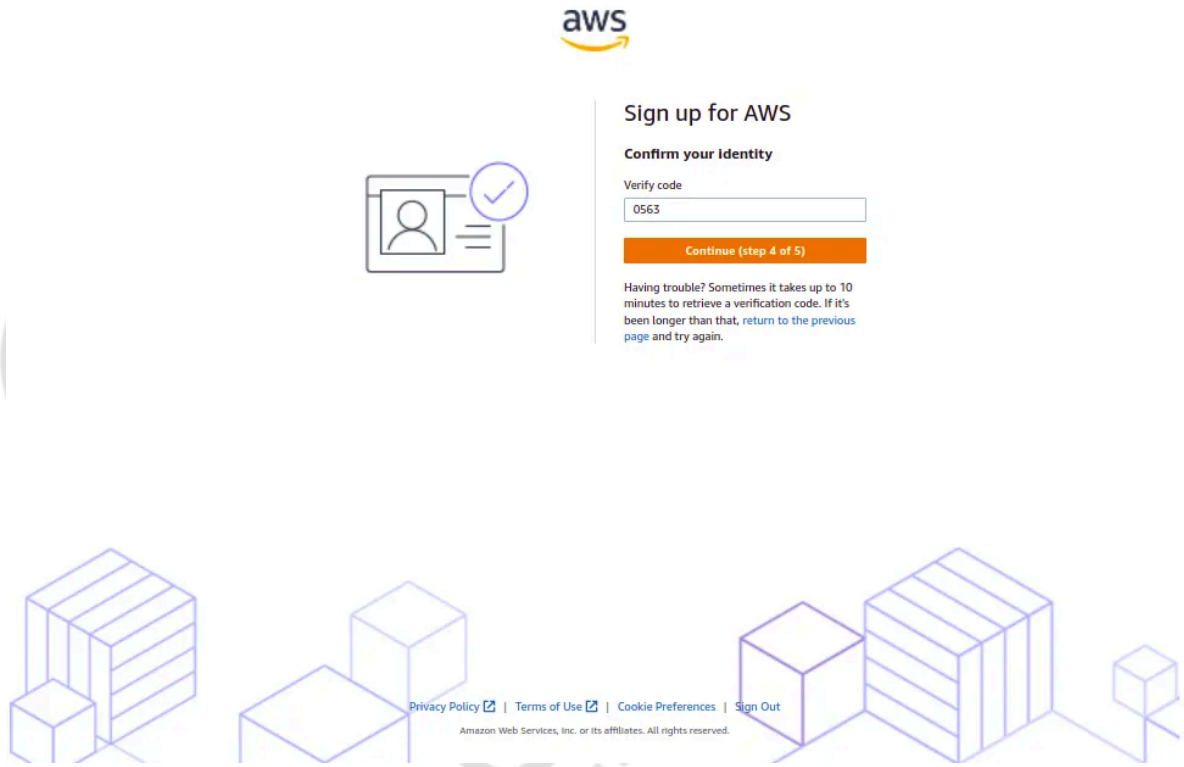
Now you have to confirm your identity to activate your account. You can choose an Text message(SMS) or a Voice call to receive the verification code.And also presented with a CAPTCHA, enter the displayed code, and then **send**.



Click on the **Send** button to continue.After the Verification send to Respective phone number.

In my case, I choose Text message(SMS). After receiving the verification code,to copy the verification code.

Fill the verification code and Click **Continue**.

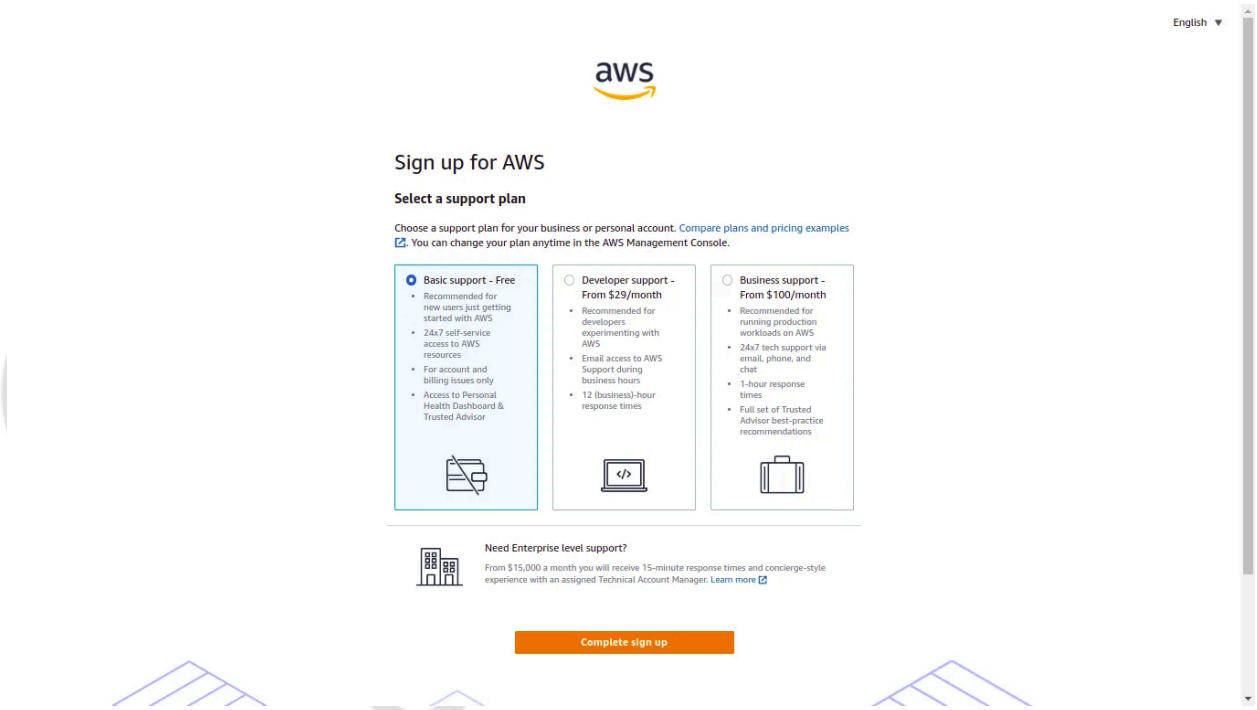


On the Select a support plan page, choose one of the available Support plans.

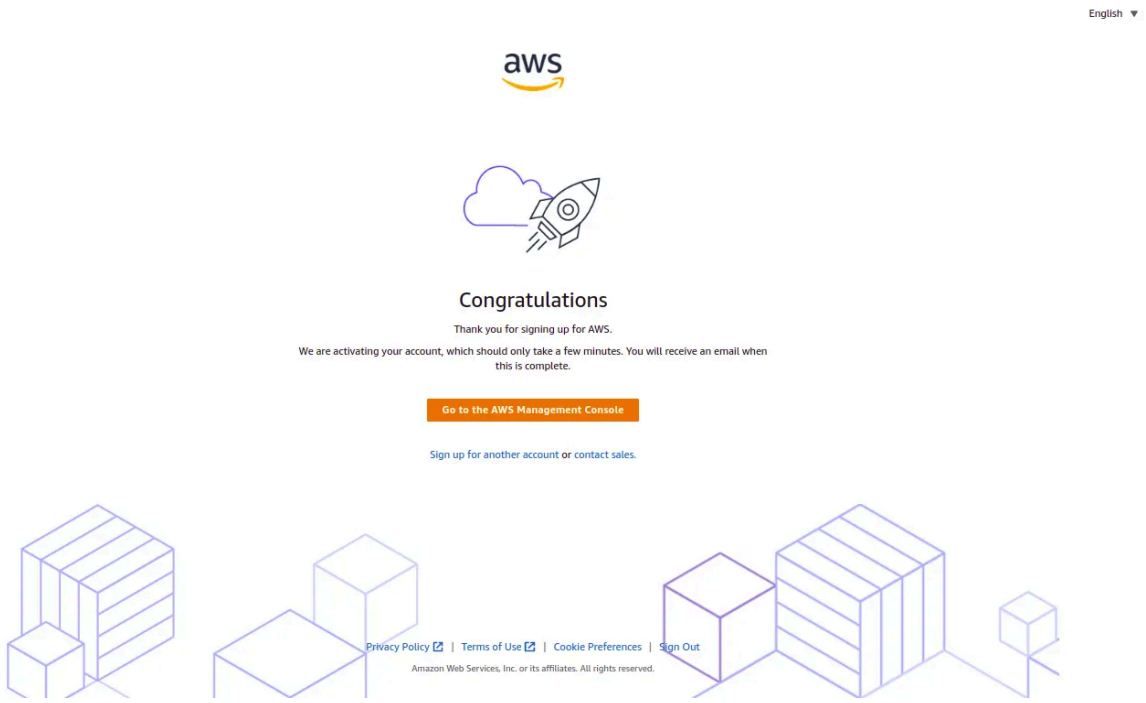
You can start with the **Basic support - Free** . I strongly recommend reading about the Support Plans to be aware of what each one covers.

In my case, I choose **Basic support - Free.**

Then, Choose **Complete sign up**.



Amazon Web Services account has successfully been created. Click **Go to the AWS Management Console**.



Git Installation and Setup

1. Download: Download the latest Git version for your operating system.
2. Install: Follow the prompts to install Git.
3. Verify Installation: Open a terminal and type git --version to verify the installation.
4. GitHub Setup: Go to GitHub.com and create an account if you haven't already.

Jenkins Installation and Setup

1. Install JDK: Download and install JDK 8 according to your system configuration. Set the path for the JDK and JRE in the system environment variables.
2. Download Jenkins: Download Jenkins for your system (e.g., Windows) and follow the installation steps.
3. Run Jenkins: Once installed, run Jenkins on localhost:8080. Use the initial admin password to log in.
4. Jenkins Interface:
   - New Item: Create a new project.
   - Build History: Track the status of your builds.
   - Manage Jenkins: Configure system settings.
5. Create a Jenkins Job:
   - Setup: Create a freestyle project.
   - Build: Add a build step to execute a batch command (e.g., echo "Hello... This is my first Jenkins Demo: %date%: %time% ").
   - Run: Click on "Build Now" to execute the job and view the console output.

Docker Installation and Setup

1. Install Docker: Install Docker Desktop on Windows or Docker for Ubuntu.
2. Verify Installation: Run docker --version to ensure Docker is installed.
3. Run Docker: Start the Docker service and ensure the Docker dashboard is running.
4. First Docker Command: Run the command docker run -d -p 80:80 docker/getting-started to get started.

This detailed document provides a comprehensive introduction to DevOps, covers its lifecycle, and guides setting up essential DevOps tools like Git, Jenkins, and Docker.

codingninjas